




OLAP 查询性能优化十问

康凯森

2023-03-28



▶ 查询性能优化十问

- 查询性能优化的意义
- 查询性能优化的目标
- 如何发现性能瓶颈点
- 如何进行性能优化
- 如何做好性能测试
- 性能优化有尽头吗?
- 生产环境和BenchMark性能有哪些差距
- 性能优化的权衡
- OLAP 查询性能优化的未来
- 如何成为 查询性能优化专家

一 查询性能优化的意义

- (产品) 性能提升 10 倍 → 用户的机器资源可以节省 十分之九
- (产品) 从 10 秒 到1秒 → 交互式分析体验
- (产品) 敲门砖：产品 POC 几乎不会缺少的环节
- (研发) 专业能力的提升：从架构到细节，从硬件到软件，从内核到应用

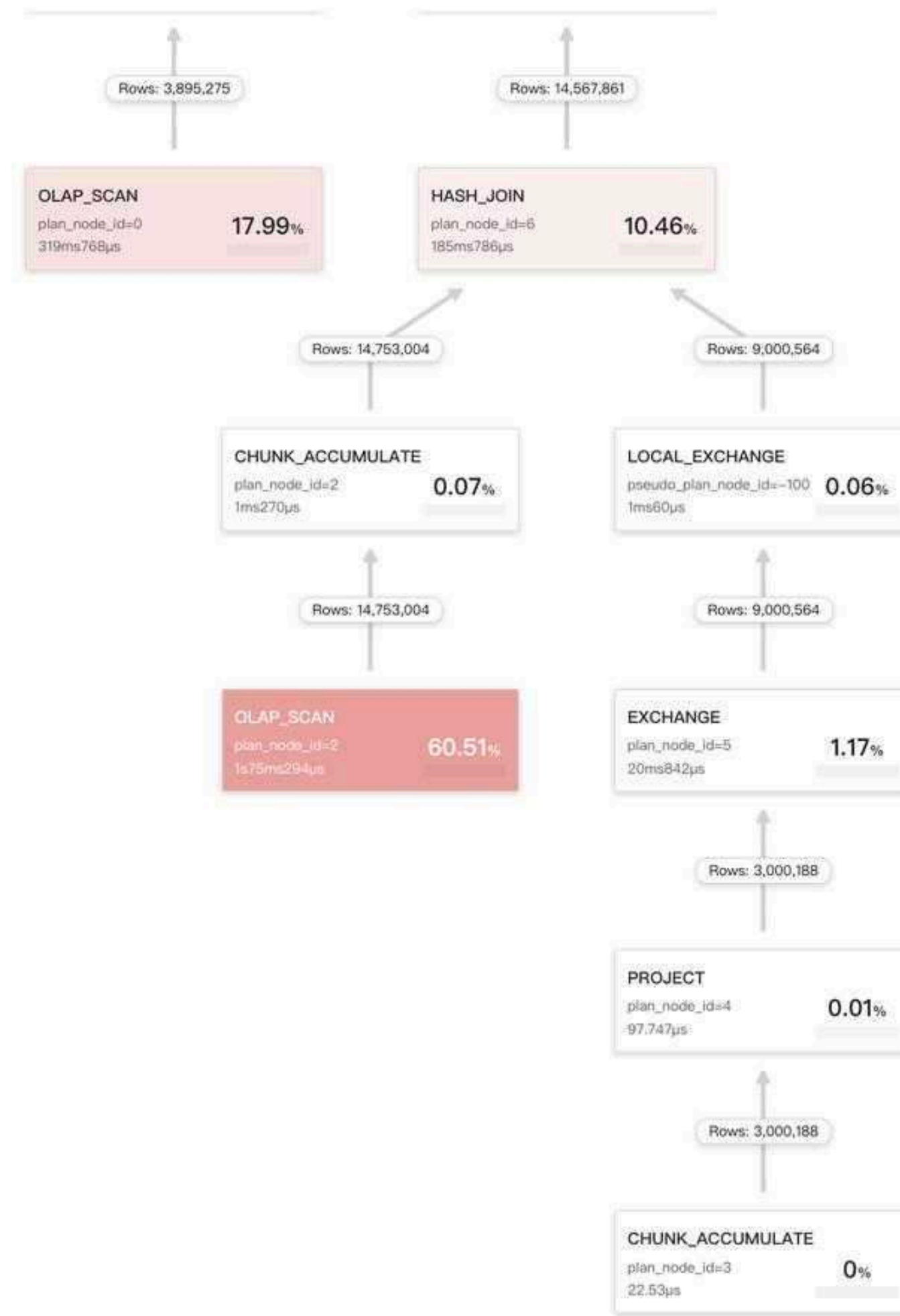
二 查询性能优化的目标

- 应用视角
 - 吞吐 Throughput
 - 时延 Latency
 - 在总资源不变的情况下 缩短响应时间，一般都可以提升吞吐率。
- 系统资源视角
 - 资源使用率（关注 查询吞吐和时延的同时，一定要关注资源使用）
 - 饱和度

▶ 三 如何发现瓶颈点

- StarRocks 自身的可观测性工具
- 数据库领域的性能测试工具
- Linux 通用的性能测试工具

三 如何发现瓶颈点 —— SR Observability : Profile



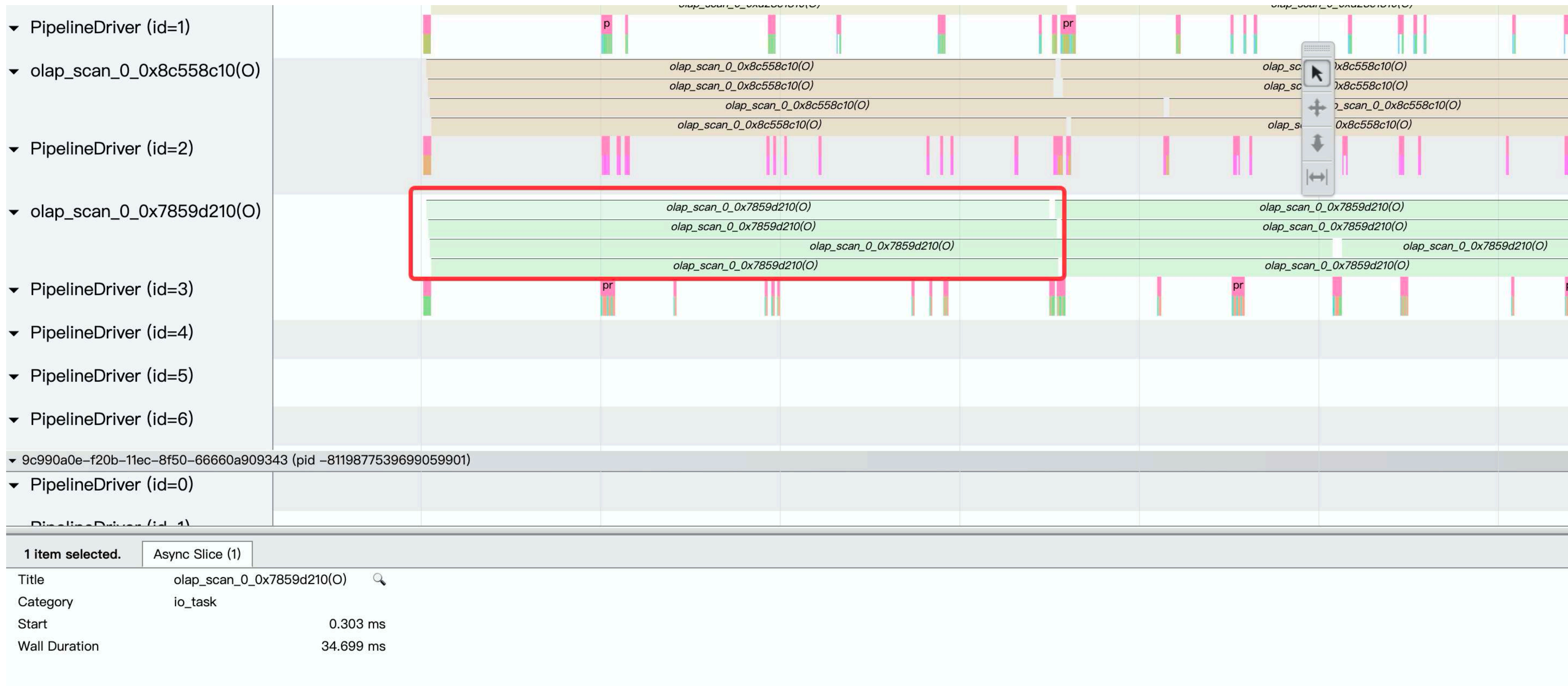
Profile Overview	
Execution time	
IO	23.61%
Processing	76.39%
ExecutionWallTime 1s496ms0µs	
IO	
DiskReadRows	488,494,519
DiskReadBytes	1.52GB
ResultRows	21,648,467
ResultBytes	4.73GB
Network	
Bytes sent over network	0Bytes

三 如何发现瓶颈点 —— SR Observability : Optimizer Trace

```
mysql> TRACE OPTIMIZER select count(distinct id_int) from test_basic;
+-----+
| Explain String |
+-----+
| 60693ms|-- Total[1] 1ms |
| 60693ms|  -- Analyzer[1] 0ms |
| 60693ms|  -- Transformer[1] 0ms |
| 60693ms|  -- Optimizer[1] 0ms |
| 60693ms|    -- Optimizer.preprocessMvs[1] 0ms |
| 60693ms|    -- Optimizer.RuleBaseOptimize[1] 0ms |
| 60693ms|      -- Optimizer.RuleBaseOptimize.RewriteTreeTask[37] 0ms |
| 60694ms|    -- Optimizer.CostBaseOptimize[1] 0ms |
| 60694ms|      -- Optimizer.CostBaseOptimize.OptimizeGroupTask[5] 0ms |
| 60694ms|      -- Optimizer.CostBaseOptimize.OptimizeExpressionTask[10] 0ms |
| 60694ms|      -- Optimizer.CostBaseOptimize.ExploreGroupTask[8] 0ms |
| 60694ms|      -- Optimizer.CostBaseOptimize.DeriveStatsTask[10] 0ms |
| 60694ms|      -- Optimizer.CostBaseOptimize.ApplyRuleTask[16] 0ms |
| 60694ms|      -- Optimizer.CostBaseOptimize.EnforceAndCostTask[10] 0ms |
| 60694ms|    -- Optimizer.PhysicalRewrite[1] 0ms |
| 60694ms|  -- ExecPlanBuild[1] 0ms |
+-----+
16 rows in set (0.00 sec)
```

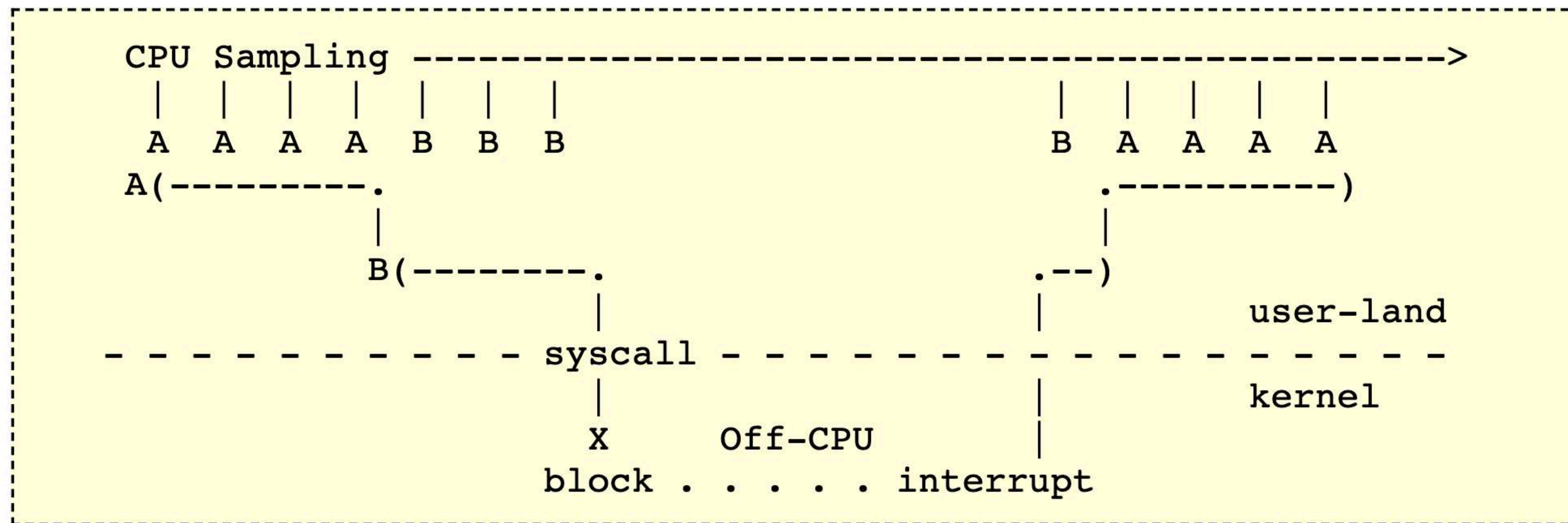
优化器

三 如何发现瓶颈点 —— SR Observability : Executor Trace



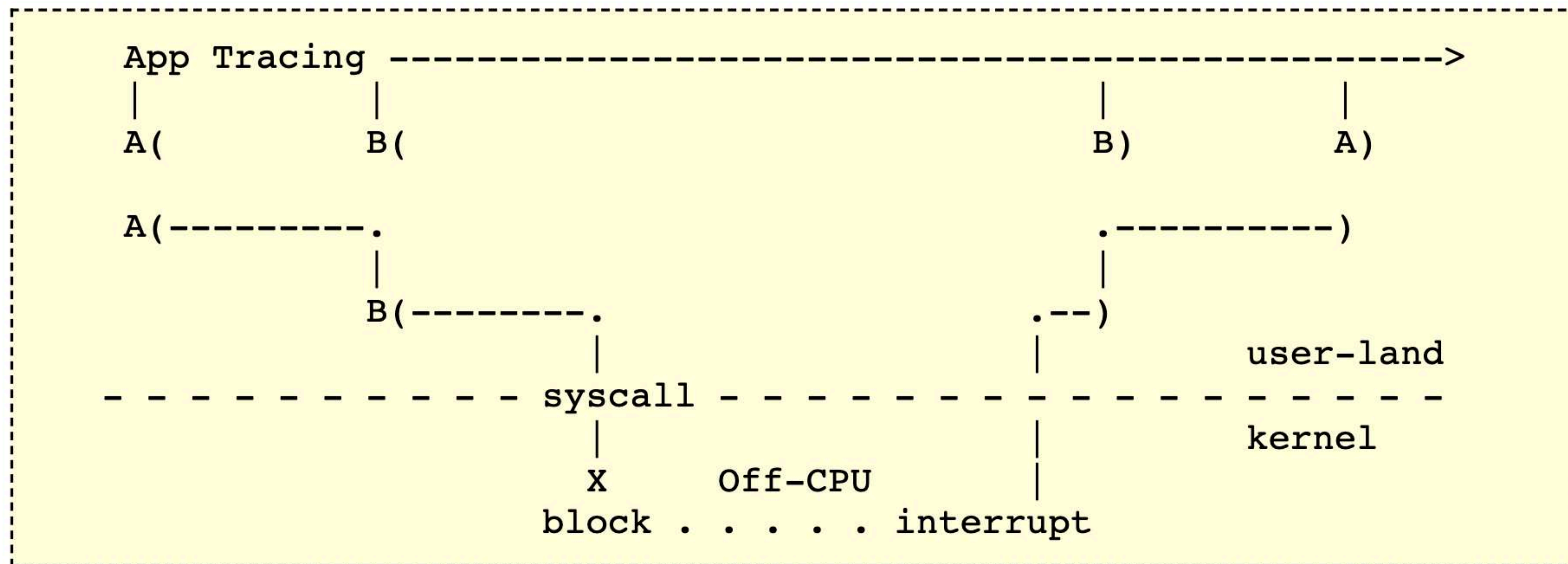
执行器

三 如何发现瓶颈点 —— On CPU Sampling (perf)



CPU 热点

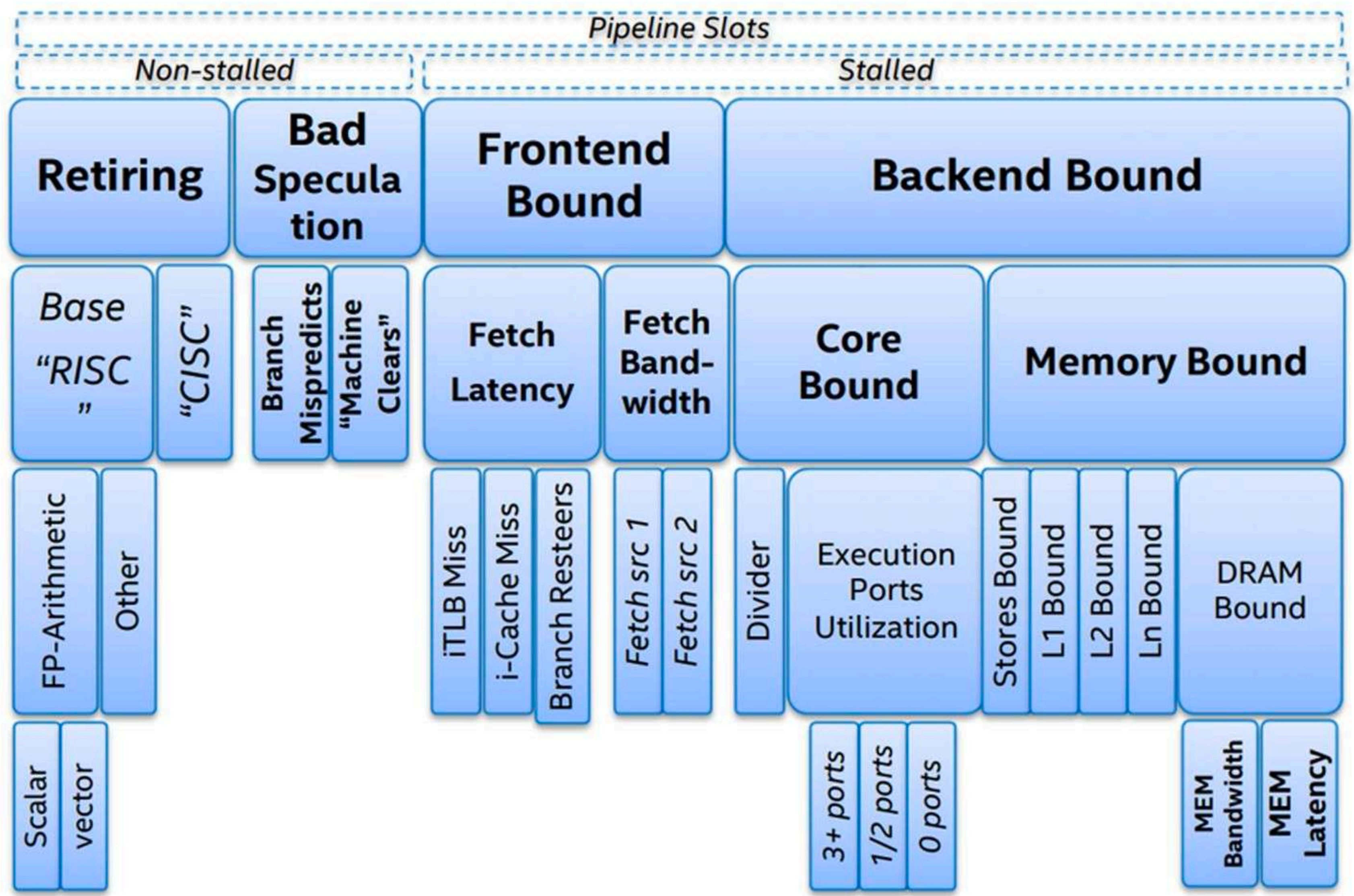
三 如何发现瓶颈点 —— Off CPU Tracing (eBPF)



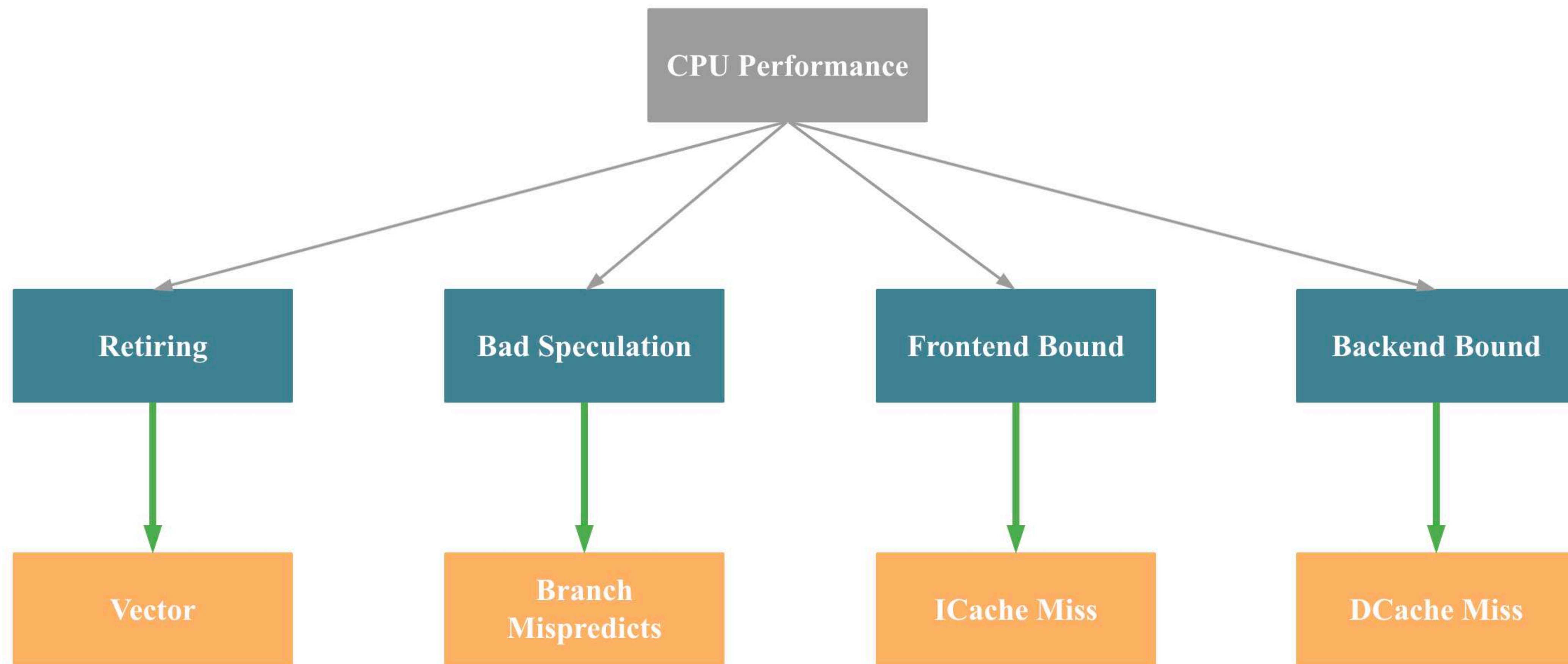
IO, 网络, Lock

<https://www.brendangregg.com/offcpuanalysis.html>

三 如何发现瓶颈点 —— TOP DOWN 分析方法

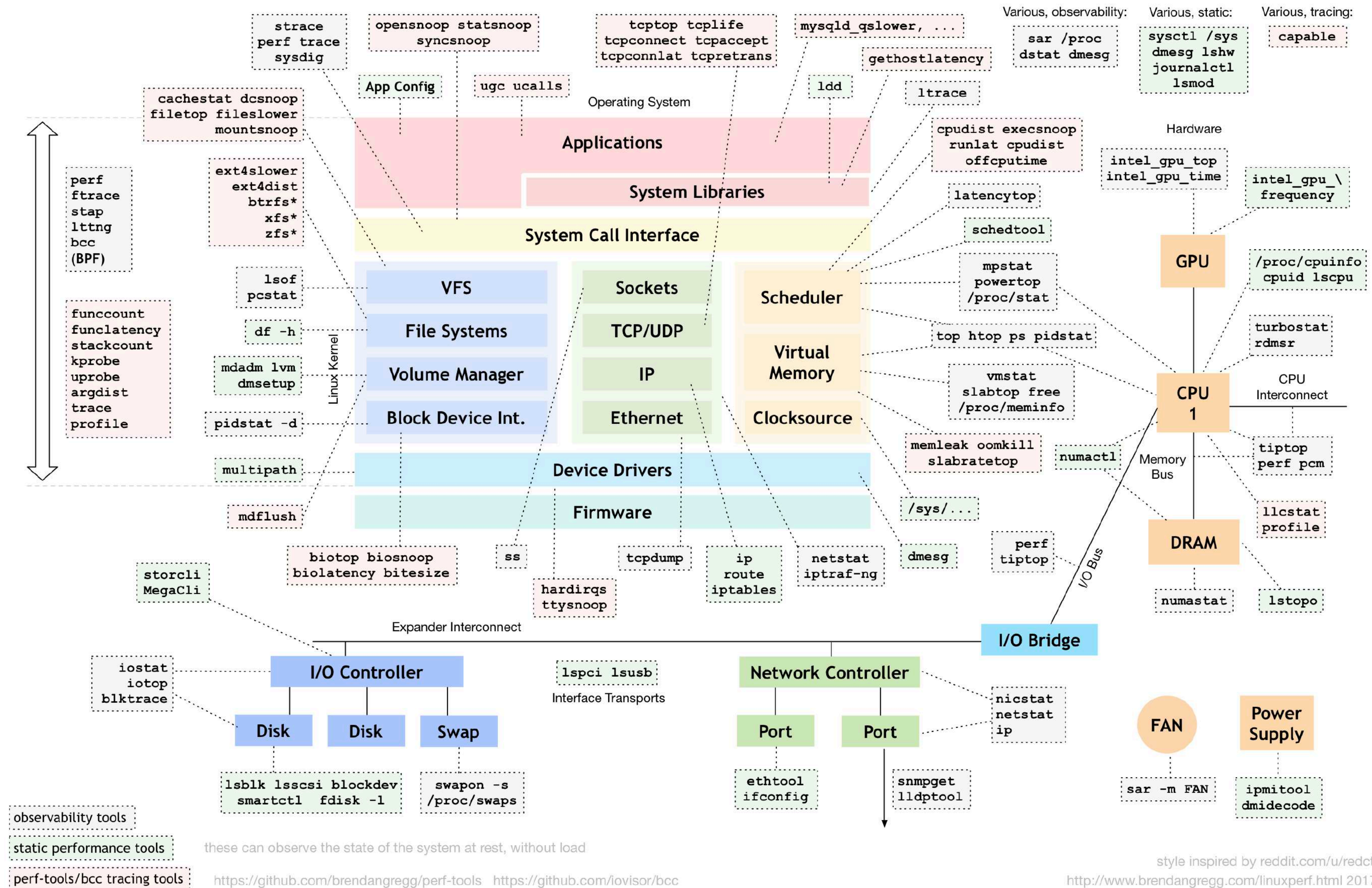


三 如何发现瓶颈点 —— TOP DOWN 分析方法

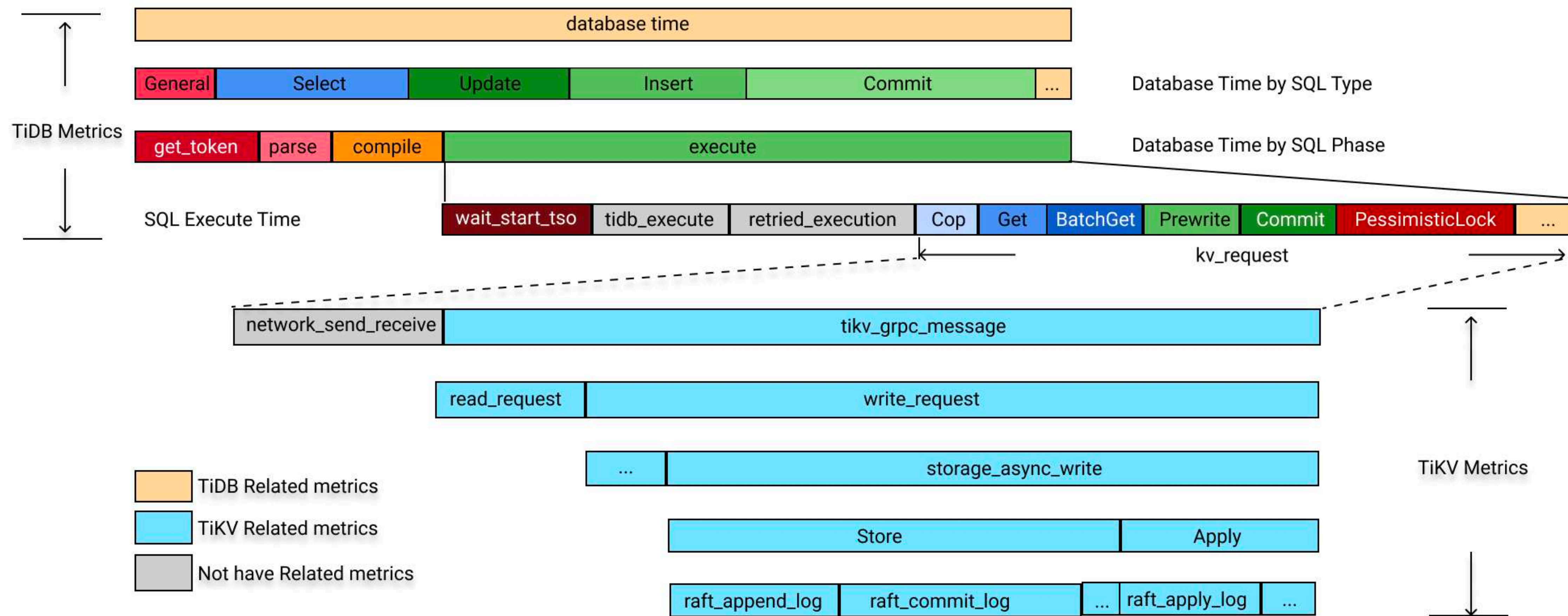


三 如何发现瓶颈点 ——— Performance tools

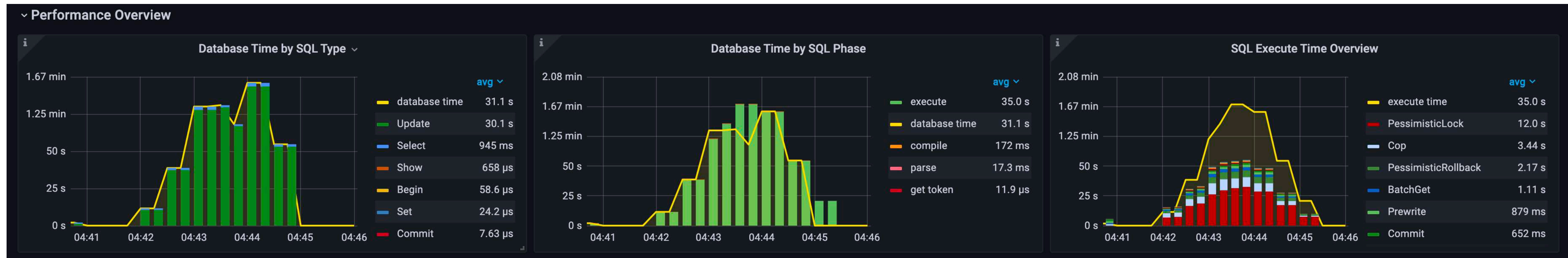
Linux Performance Tools



三 如何发现瓶颈点——性能指标监控



三 如何发现瓶颈点 —— 性能指标监控



Update 类型的SQL block 在悲观锁上

<https://docs.pingcap.com/tidb/dev/performance-tuning-methods>

三 如何发现瓶颈点 —— 优化器 Plan Test

/* [First query, 75 milliseconds] */

```
SELECT Max(emp.sal)
FROM dept INNER JOIN emp ON ename NOT LIKE name
WHERE name = 'ACCT';
```



ename NOT LIKE 'ACCT'

/* [Second query, 238 milliseconds] */

```
SELECT Max(emp.sal)
FROM dept INNER JOIN emp ON ename NOT LIKE name
WHERE name = 'ACCT' IS TRUE;
```

~~ename NOT LIKE 'ACCT'~~

/* [First query, 25 milliseconds] */

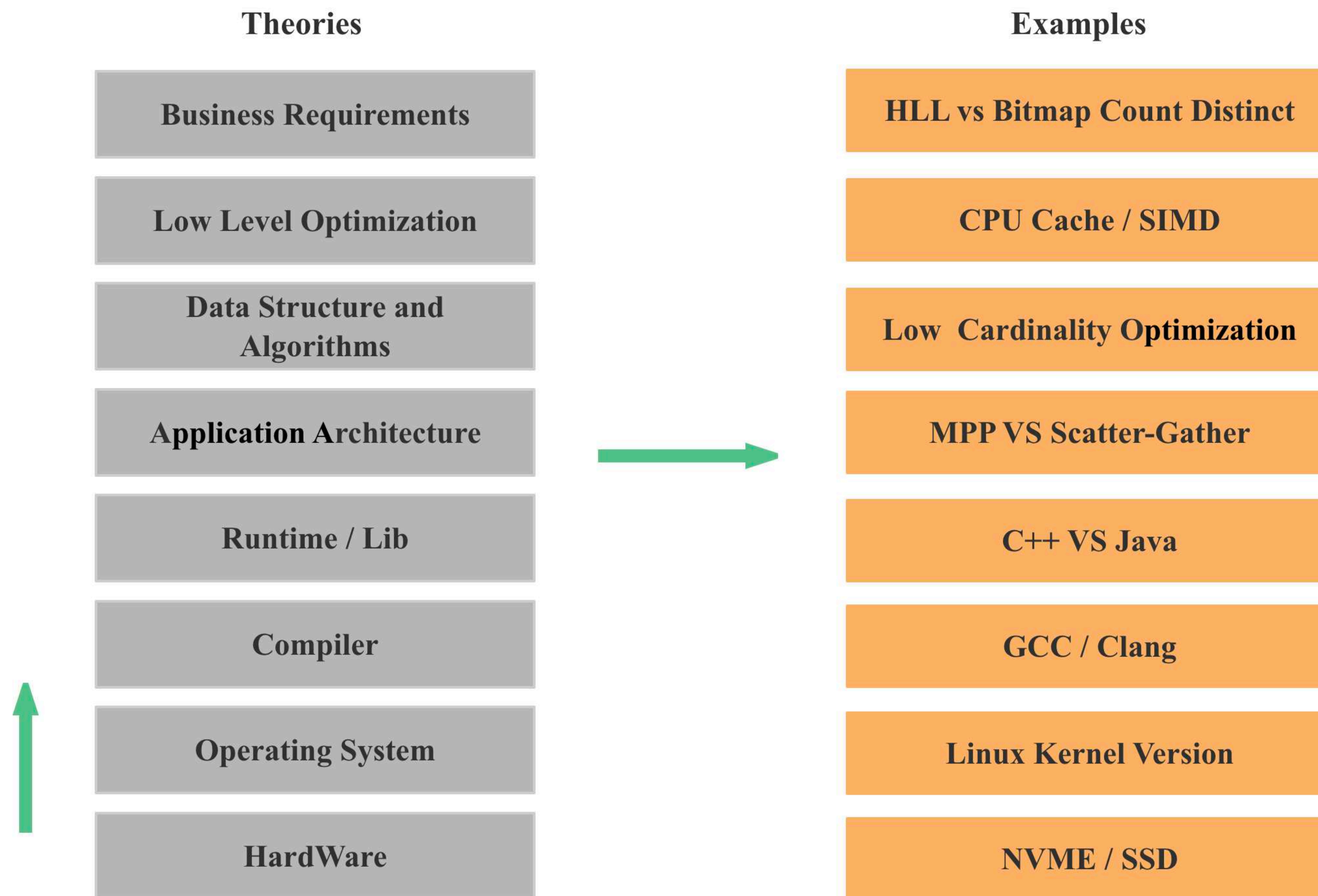
```
SELECT emp_pk FROM emp WHERE emp_pk > 100;
```

emp_pk 是主键

/* [Second query, 72 milliseconds] */

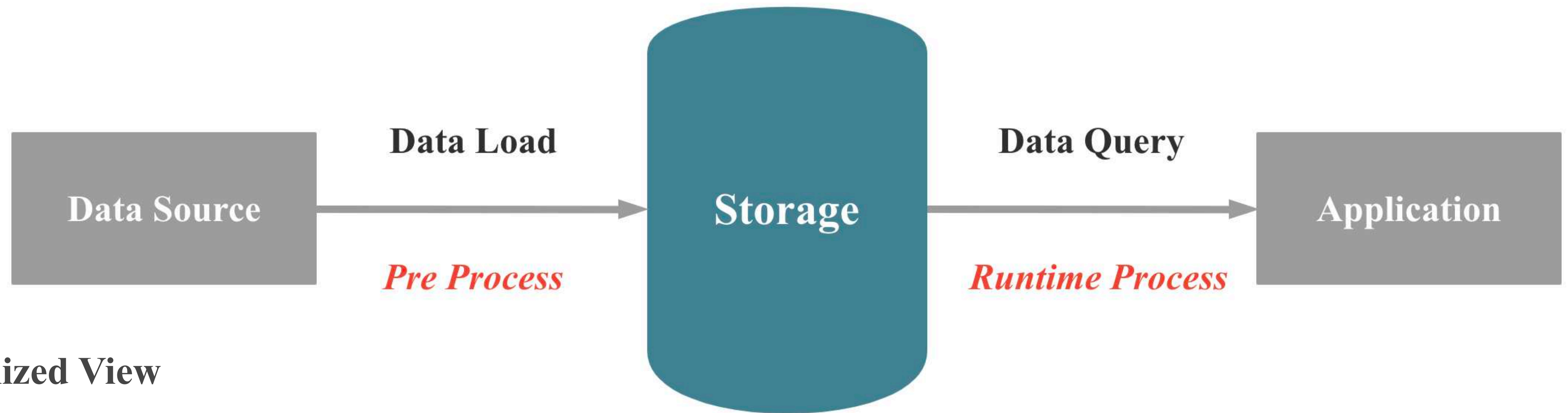
```
SELECT emp_pk FROM emp WHERE emp_pk > 100 GROUP BY
emp_pk;
```


四 How 性能优化：CPU 通用性能优化



数据库只是一个大型的 CPU 应用

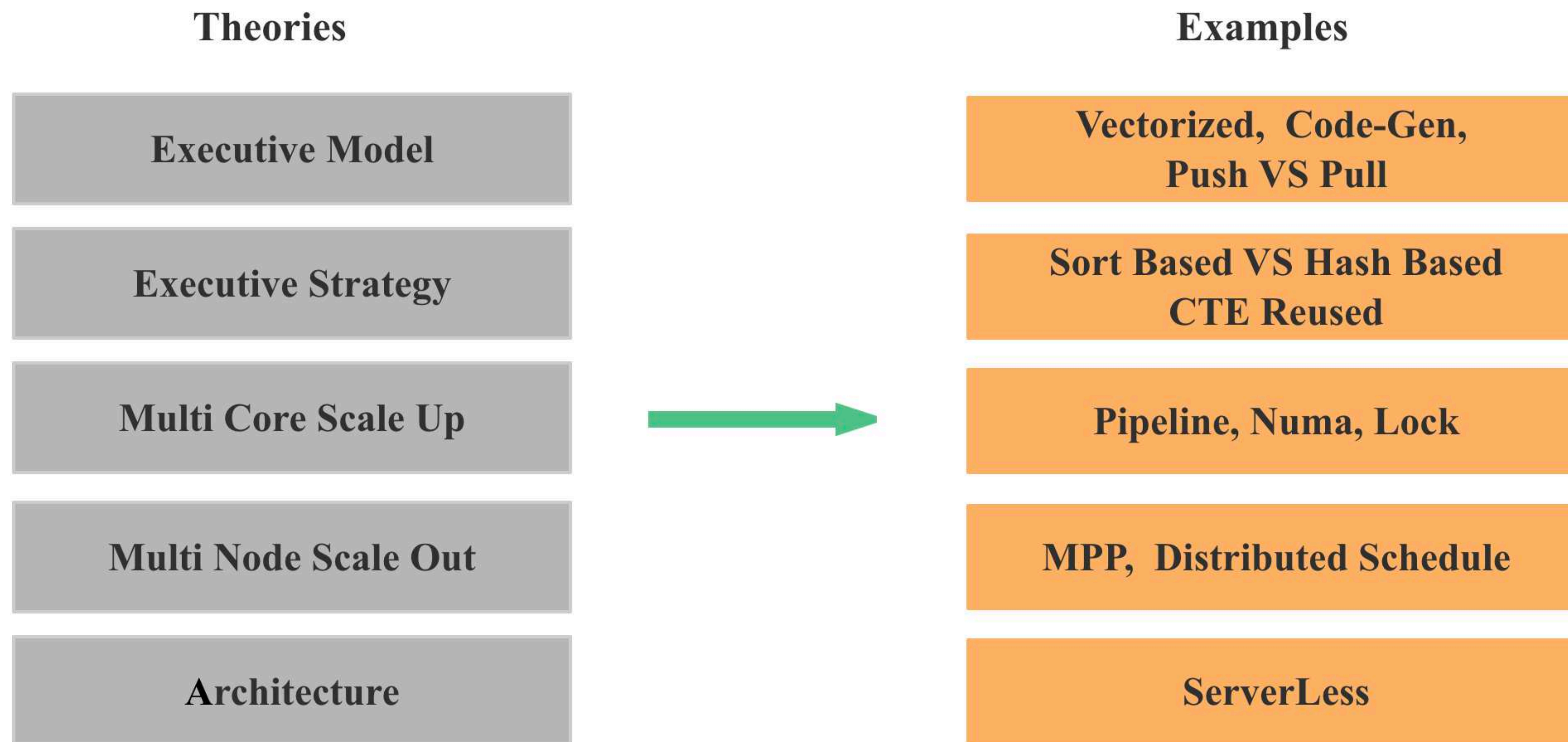
四 How 性能优化: DataBase Pre Process VS Runtime Process



- Materialized View
- Aggregate Data When Load
- Index
- Cache

The more pre process, The less runtime process

四 How 性能优化: DataBase High Level Optimization



四 How 性能优化: DataBase Low Level Optimization

C++ Language Optimization

Memory Management

Branch Mispredicts

CPU Cache

SIMD

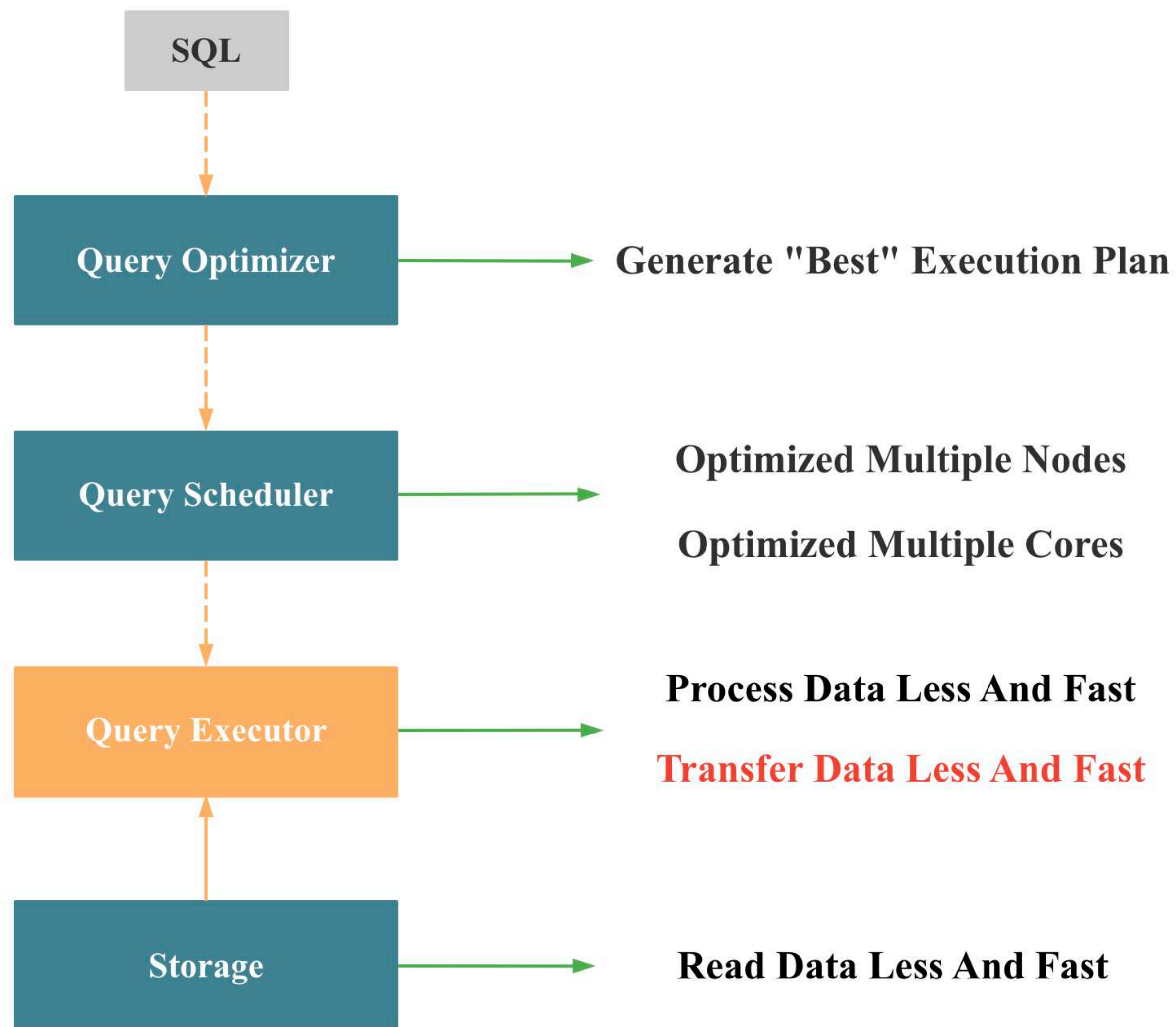
Data Structure and Algorithms

▶ 四 How 性能优化：资源角度

- Read Data Less And Fast (**IO**)
- Transfer Data Less And Fast (**Network**)
- Process Data Less And Fast (**CPU & Memory**)

四 How 性能优化：资源角度（网络）

- Shuffle By Column
- Compress Data By Column
- Global Runtime Filter
- Operations On Encoded Data
- Colocate Join
- Replication Join
- Bucket Shuffle Join



四 How 性能优化：ClickBench 优化总结

- 性能工具
- 常见优化思路
- 等价变换
- 原理的深刻理解

优化	优化层次	优化点	优化组件	资源视角	SR PR
全表 Max Min 走 Meta Scan	High	执行策略	优化器	IO	15542
全表 Count 走 Meta Scan	High	执行策略	优化器	IO	17258
优化 Cast 表达式的 nullable 属性	High	不做无用功	优化器	CPU & Memory	15380
sum(col), sum(col + 1) => sum(col), sum(col) + count() * 1	High	执行策略	优化器	CPU	16520
group by a, a + 1 => group by a	High	不做无用功	优化器	CPU	16716
Project 表达式和聚合算子融合	Low	Cache Miss	优化器 & 执行器	CPU	15998
TopN Runtime Filter	High	执行策略	执行器	IO	15949
优化 regexp_replace 函数性能	Low	内存复用	执行器	Memory	16356
优化字符串类型的PlainPage读取	Low	C++ 优化	执行器	CPU & Memory	16519
Remove lock for CompressionContextPool	Low	C++ 优化 Scale up	执行器	CPU	16708

五 如何做好性能测试

- 对比测试：硬件环境，数据，建模等基础信息对齐 (出过很多次问题)
- 不同硬件 (核数多少，磁盘介质，出过多次问题)
- 不能只关注单并发，还要关注高并发
- 不能只关注延迟和吞吐，还要关注资源利用率
- 不能只关注 AVG，还要关注PT99和抖动
- 不能只关注目标场景，各种典型 workload 的查询都要测试
- 细致，周密，敏锐，自动化，标准化

▶ 六 CPU 架构下的数据库性能优化有尽头吗？

- 硬件在持续变化
- 数据库架构在持续变化
- 更多的上下文，更多的优化策略
- 数据结构和算法层面的创新
- 执行策略层面的创新
- From Manually To Adaptive

性能优化永无止境

七 生产环境的性能 VS Benchmark 的性能

- 大查询影响小查询
- 导入，查询，Compaction，统计信息等任务相互影响
- 并发控制 & 查询排队
- 查询超时 & Retry
- PT99, Not Avg
- 慢节点
- 数据倾斜

生产环境处处充满挑战

八 性能优化的权衡

- 代码复杂度
- 兼容性
- 稳定性
- 优化的投入产出比
- 优化的通用性
- 性能的可预测性

不是所有的需求都要满足

不是所有的Bug都要修复

不是所有的优化都要实现

九 OLAP 数据库性能优化的未来

- Serverless 架构下面向成本的性能优化: $10 \text{ core} * 10 \text{ s} = 100 \text{ core} * 1 \text{ s}$
 - 避免单点，避免串行，避免数据倾斜
- 真实业务场景的自适应优化
 - 数据分布，基数，相关性，数据倾斜
- 真实历史数据的 AI 优化
 - 统计信息，资源使用，并行度

十 如何成长为数据库性能优化专家

- CPU & 内存 & 网络 & IO 的专业知识 (原理, 性能指标, 性能工具)
- 数据库领域的专业知识
- 性能测试的工具和方法论
- 数据库领域各种优化思路,
- 关注学术和工业界的进展
- 关注新硬件, 新架构
- 目标系统原理, 源码的深刻理解和掌握

实践出真知, 多动手, 多尝试

思考：AGI时代，数据库性能优化将如何改变？

AGI时代，AI能做什么，不能做什么

AGI时代，什么将是我们的核心竞争力

AGI时代，我们如何利用好AI的能力

附录：写入时优化 VS 查询时优化 VS Compaction 时优化

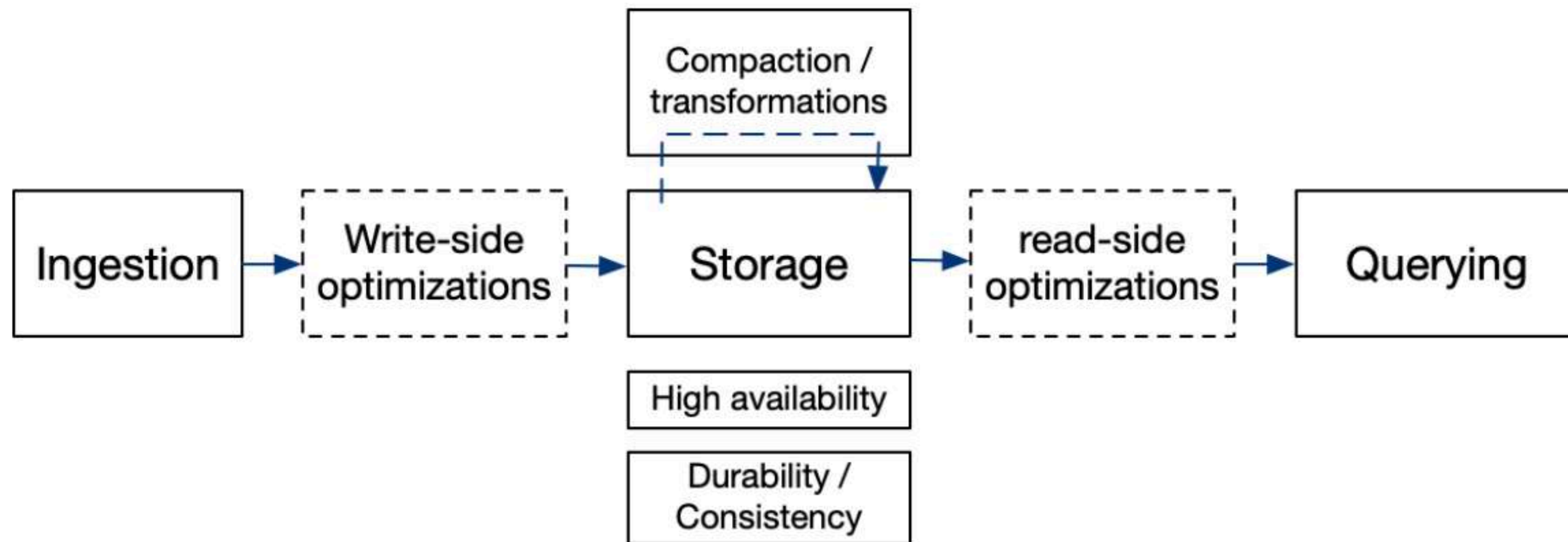
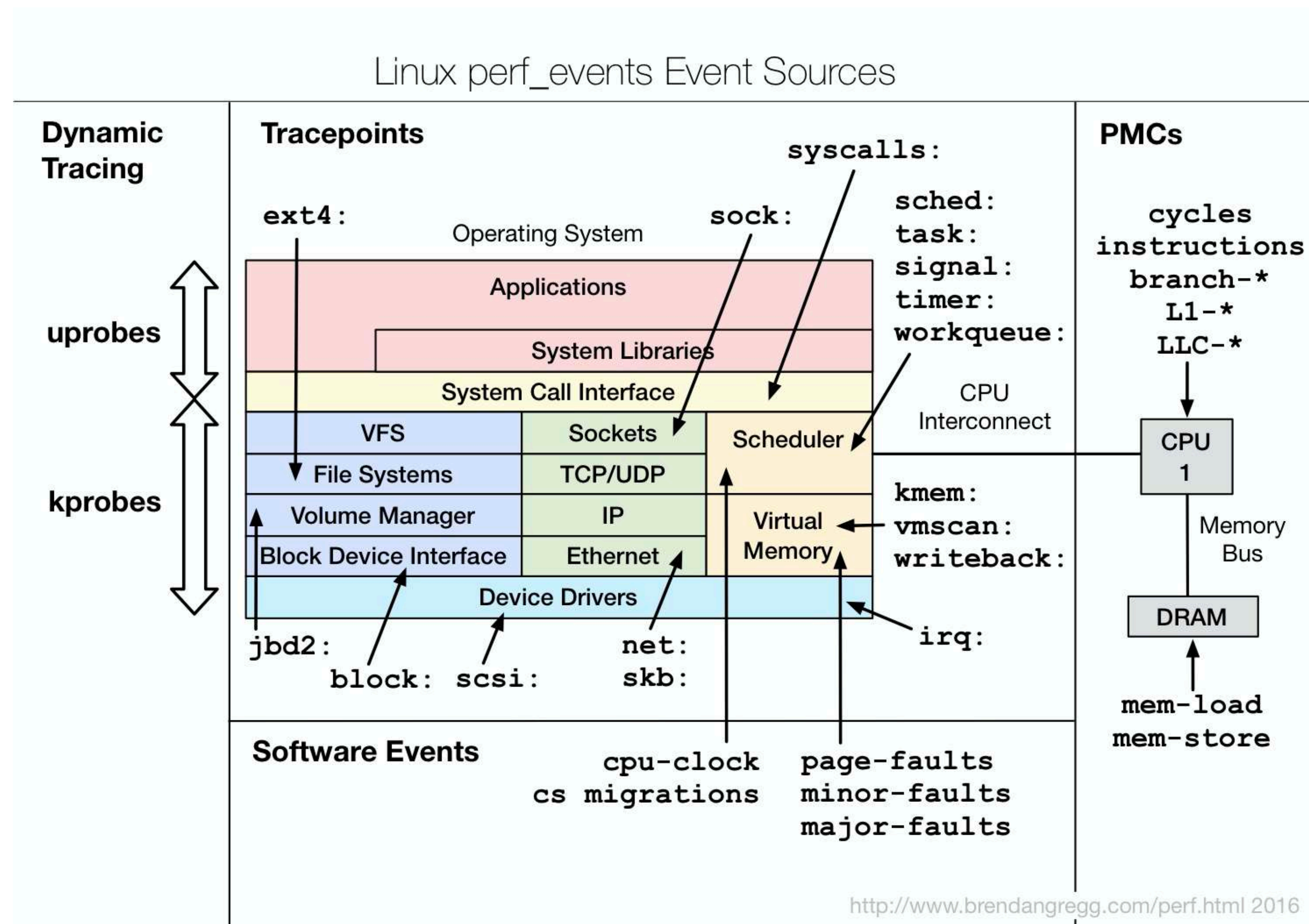


Figure 2: High level components for analytical systems

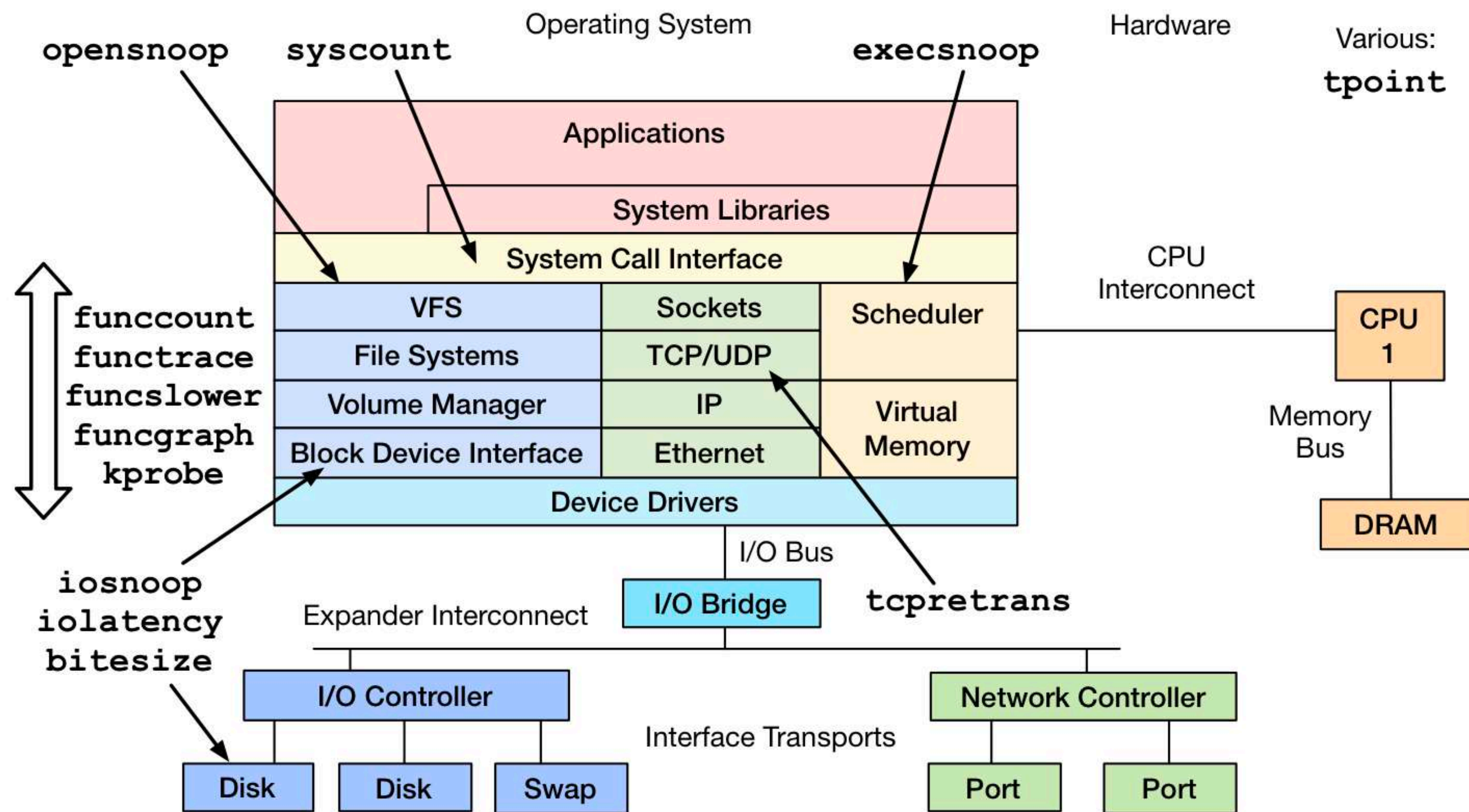
附录：Linux perf_event Event Sources



<https://www.brendangregg.com/perf.html>

附录：Perf-tools

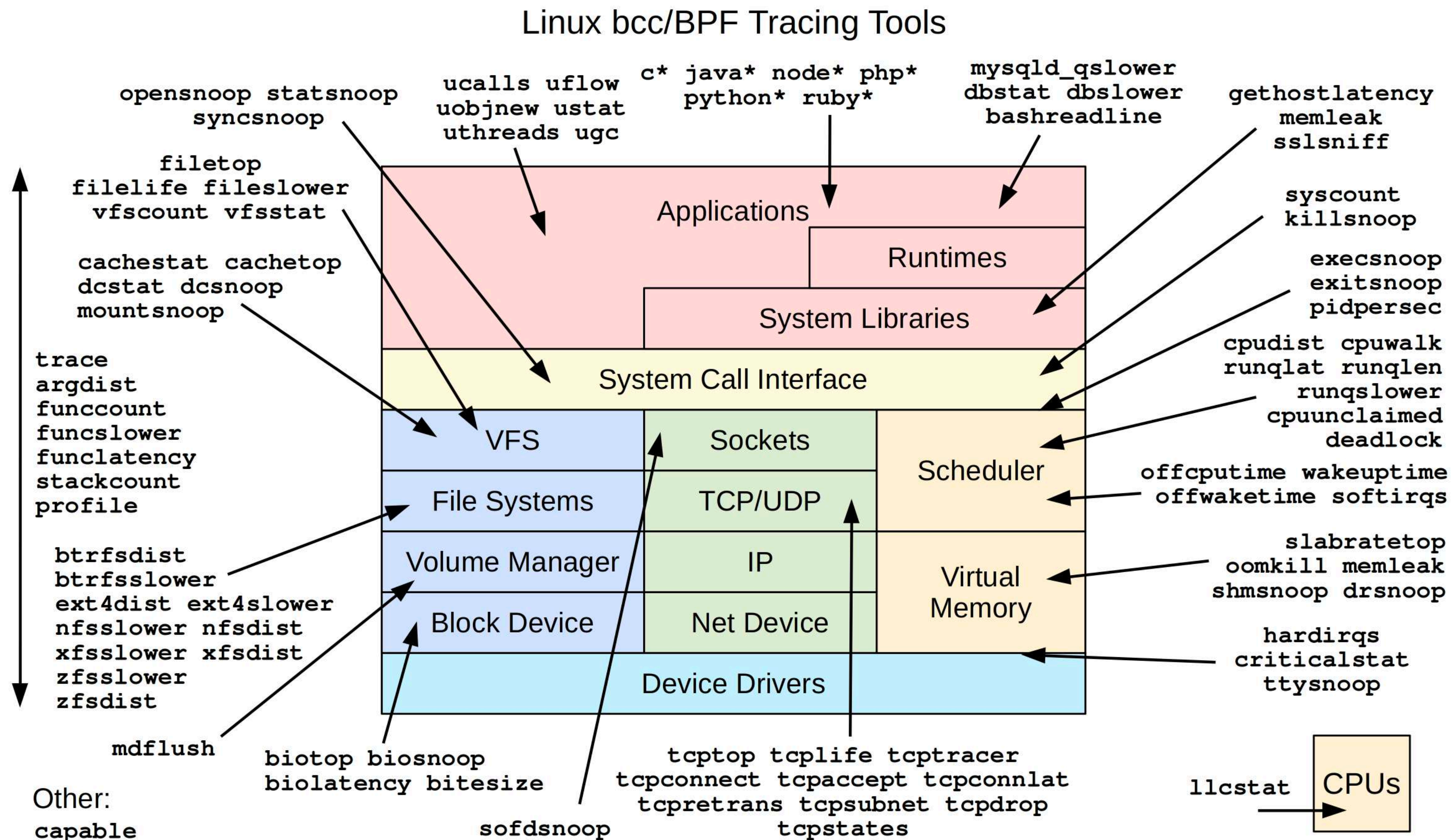
Linux Performance Observability Tools: perf-tools



<https://github.com/brendangregg/perf-tools#contents>

<https://github.com/brendangregg/perf-tools>

附录：bcc tracing tools



<https://github.com/iovisor/bcc#tools> 2019

<https://github.com/iovisor/bcc>

2023

Thanks